4CS4-6: Theory of Computation (NP, NP-complete, NP-Hard)

Lecture 26: April, 23, 2020

Prof. K.R. Chowdhary *: Professor of CS*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

26.1 The Class NP

Definition 26.1 NP. *A language is in* **NP** *iff it is decided by some Nondeterministic Turing Machine (NDTM) in Polynomial time. An NDTM guesses the alternatives.*

Polynomial solution for these problems are not known to exist. In an NDTM the solution is selected nondeterministically rather than systematically examining all the possibilities. Therefore $\mathbf{P} \subseteq \mathbf{NP}$, because a **P** problem is also an **NP** (see Fig. 26.1). Hence, **NP** is class of membership problems for languages in,

$$\mathbf{NP} = \bigcup_{P(n)} NTIME(P(n)). \tag{26.1}$$

Following the examples of **NP** problems:

1. *SATISFIABILITY Problem.* Input is Boolean expression u in CNF (Conjunctive Normal Form), and output is Yes, if there is an assignment that satisfies u else No. The Complexity in **P** is unknown, while in **NP** is Yes. This problem is also called SAT problem.

   Definition 26.2 *NP-Complete. A decision problem B is NP-complete if:*

   (a) B is in NP, and

   (b) every problem A in NP is reducible to B in polynomial time.

   The problem B can be shown to be in NP by demonstrating that a candidate solution to B can be verified in polynomial time.

   A consequence of this definition is that if we had a polynomial time algorithm on a Universal TM (UTM), for B, we could solve all problems in NP in polynomial time.

   The Satisfiability problem is NP-complete. A Boolean expression $\phi = \{\bar{x} \vee y) \wedge (x \vee \bar{z})$ is satisfiable for $x = 0$, $y =$ don't care, $z = 1$, as it evaluates ϕ to 1 (TRUE).

   SAT is languages of all satisfiable formulas,

$$\mathbf{SAT} = \{\langle \phi \rangle \mid \phi \text{ is satisfiable Boolean formula}\}. \tag{26.2}$$

Cook-Levin theorem links the complexity of SAT problem to complexities of all problems in NP. Using this, any NP problem can be converted into SAT in polynomial time.

2. *Hamiltonian Path Problem.* The input is directed graph $G = (V, E)$, and output is Yes if there is a single cycle that visits all nodes, and No other wise. The complexity **P** is unknown, and **NP** is Yes. A Hamiltonian path problem is in NP, but its solution can be verified in **P**.

3. *Subset Sum Problem.* The Input is set S, number k, and output is YES if there is set $P \subseteq S$, whose total is k, else No. The Complexity class **P** is unknown, while **NP** is YES.

The table 26.1 shows the various complexity classes for Time.

Table 26.1: Complexity classes for Time

Class	*Machine*	*Time constraint*
DTIME(f(n))	DTM	$f(n)$
P	DTM	$poly(n)$
NTIME(f(n))	$NDTM$	$f(n)$
NP	$NDTM$	$poly(n)$
EXPTIME	DTM	$2^{poly(n)}$

26.2 Space Complexity

Definition 26.3 S(n). *A function $S(n)$ is called space constructible if there exists an $S(n)$ space-binded Deterministic Turing Machine that, for each input $|w| = n$ requires exactly $S(n)$ space. Therefore, $S(n)$ is space complexity of a deterministic Turing Machine.*

Definition 26.4 DSPACE(S(n)). *It is class of languages that have space complexity of $O(S(n))$, on deterministic TM.*

$$DSPACE(S(n)) = \{L \mid L \text{ is decidable by } O(S(n)) \text{ space on } DTM\}.$$

Definition 26.5 PSPACE : *The class of membership problems for the languages decidable in polynomial space on deterministic Turing Machine are:*

$$PSPACE = \bigcup_k DSPACE(n^k) \tag{26.3}$$

Definition 26.6 *$NSPACE(S(n))$:*

$$NSPACE(S(n)) = \{L \mid L \textit{ is decidable by } O(S(n)) \textit{ space on } NDTM\}$$

The table 26.2 shows the details of various space complexities.

Table 26.2: Space Complexities

Class	*Machine*	*Space constraint*
DSPACE(f(n))	DTM	$f(n)$
L	DTM	$O(\log\ n)$
PSPACE	DTM	$poly(n)$
EXPSPACE	DTM	$2^{poly(n)}$
NSPACE(f(n))	$NDTM$	$f(n)$
NL	$NDTM$	$poly(n)$
NEXPSPACE	$NDTM$	$2^{poly(n)}$

Theorem 26.7 *Savitch's Theorem: If a Nondeterministic TM uses $f(n)$ space, it can be converted into a Deterministic TM that uses $f^2(n)$ space.*

As per Savitch's theorem: $PSPACE = NSPACE$, $EXPSPACE = NEXPSPACE$.

For nondeterinistic TM, if $f(n)$ is maximum number of tape-cells scan in any branch of computation, then its complexity is $f(n)$.

The SAT problem, which is $NP - Complete$ in time, is linear space, because the space is reusable.

$$PSPACE = NSPACE,$$

$$P \subseteq PSPACE.$$

$$NP \subseteq NSPACE,$$

Therefore, $NP \subseteq PSPACE$.

From above,

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME.$$

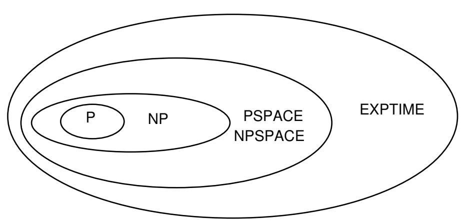

Figure 26.1: Complexity classes' relations

26.3 NP-Complete and NP-Hard Problems

Definition 26.8 NP-Complete: *A language B is NP-complete if it satisfies two-conditions: (1) $B \in NP$, (2) Every $A \in NP$ is polynomial time reducible to B, i.e.,*

$$B \in NP \wedge \forall A : A \in NP \wedge A \leq_P B \Rightarrow B \in NP\text{-}Complete.$$

Definition 26.9 NP-Hard. *A language Q is* NP-hard *if every $L \in NP$ is polynomially reducible to Q, i.e.,*

$$\forall L : L \in NP \wedge L \leq_P Q \Rightarrow Q \in NP\text{-}hard$$

.

The NP-hard problem that is also NP is called NP-complete. The figure 26.2 shows the relations between P, NP, NP-complete, and NP-hard problems for $P \neq NP$.

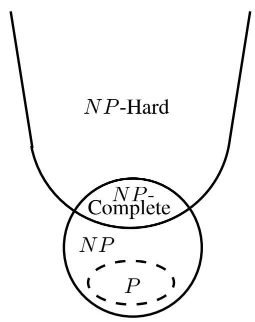

Figure 26.2: P, NP, NP-complete, and NP-hard set of problems

Definition 26.10 Co-NP: *Co-NP is complement of NP, therefore, Co-NP is set of all the complements of all the NP problems.*

Theorem 26.11 NP-Complete Theorem: *If B is NP-Complete and $B \in P$, then $P = NP$.*

Proof: If B is NP-Complete then every problem in NP is polynomially reducible to B. Since $B \in P$, therefore, every NP problem is polynomially reducible to B, which is P. Hence, every NP is P, i.e. $P = NP$. ■

Once we get NP-Complete, other NP problems can be reduced to it. However, establishing first NP-Complete problem is difficult.

Theorem 26.12 *If $B \in$ NP-Complete and $B \leq_P C$ for $C \in NP$, then $C \in$ NP-Complete.*

Proof: We must show that every $A \in NP$ is polynomially reducible to C. Because B is NP-Complete, therefore, every $A \in NP$ is polynomially reducible to B. (as per property of NP-Complete). And B in turn is polynomially reducible to C (given).

Because the property of polynomial is closed under the composition, we conclude that every $A \in NP$ is polynomially reducible to C. Therefore C is NP-Complete. ■

Theorem 26.13 Cook-Levin Theorem: *SAT is NP-Complete.*

Proof Idea: It is easy to show that SAT is NP, the hard part is to show that any language in NP is polynomially reducible to SAT.

Therefore, we construct a polynomial time reduction for every $A \in NP$ to SAT. Reduction for a language A takes input w and produces Boolean formula ϕ that simulates the NP machine for A on input w.

If machine accepts, ϕ has a satisfying assignment, that corresponds to accepting computation, otherwise NO.

Therefore, $w \in A$ iff ϕ is satisfiable.

The examples of NP-Complete problems are: 3-SAT, *Hamiltonian path problem*, *subset construction problem.*

Following are also the NP-complete problems.

Boolean satisfiability problem (SAT)

Hamiltonian path problem

Travelling salesman problem (decision version)

Subset sum problem

Vertex cover problem

Graph coloring problem